Compilers: Theory, tools, scope.

Prof. (Dr.) K.R. Chowdhary, Director SETG
Email: kr.chowdhary@jietjodhpur.com
Webpage: http://www.krchowdhary.com

Jodhpur Institute of Engineering and Technology, SETG

August 22, 2014

Outlines

- Overview and History
- What Do Compilers Do?
- The Structure of a Compiler
- The Syntax and Semantics of Programming Languages
- Compiler Design and Programming Language Design
- Computer Architecture and Compiler Design
- Compiler Design Considerations

Overview and History (1)

Cause

- Software for early computers was written in assembly language
- The benefits of reusing software on different CPUs started to become significantly greater than the cost of writing a compiler

The first real compiler

- FORTRAN compilers of the late 1950s
- 18 person-years to build

Overview and History (2)

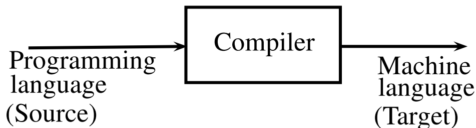
Compiler technology

is more broadly applicable and has been employed in rather unexpected areas.

1. Text-formatting languages,
2. like nroff and troff; preprocessor packages like eqn, tbl, pic
3. Silicon compiler for the creation of VLSI circuits
4. Command languages of OS
5. Query languages of Database systems

What Do Compilers Do (1)

- A compiler acts as a translator,
- transforming human-oriented programming languages
- into computer-oriented machine languages.
- Ignore machine-dependent details for programmer



What Do Compilers Do (2)

Compilers may generate three types of code:

Pure Machine Code

- Machine instruction set without assuming the existence of any operating system or library.
- Mostly being OS or embedded applications.

Augmented Machine Code

- Code with OS routines and runtime support routines. More often

Virtual Machine Code

- Virtual instructions, can be run on any architecture with a virtual machine
- interpreter or a just-in-time compiler
- Ex. Java

What Do Compilers Do (3)

- Another way that compilers differ from one another is in the format of the target machine code they generate:

  Assembly or other source format

  Relocatable binary

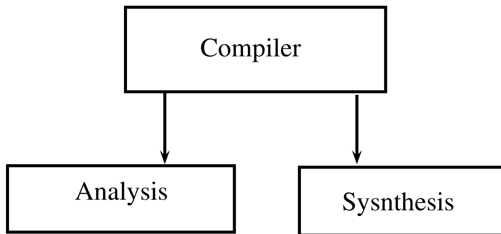
  - Relative address
  - A linkage step is required

  Absolute binary

  - Absolute address
  - Can be executed directly

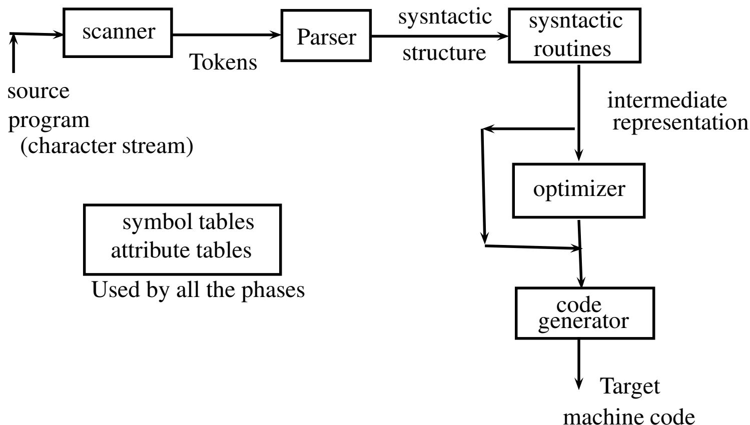
The Structure of a Compiler (1)

- Any compiler must perform two major tasks:



- Analysis of the source program
- Synthesis of a machine-language program

The Structure of a Compiler (2)



Scanner:

- The scanner begins the analysis of the source program by reading the input, character by character, and grouping characters into individual words and symbols (tokens)
  - RE (Regular expression)
  - NFA (Non-deterministic Finite Automata)
  - DFA (Deterministic Finite Automata)
  - LEX

The Structure of a Compiler: Parser

Parser:

- Given a formal syntax specification (typically as a context-free grammar [CFG]), the parse reads tokens and groups them into units as specified by the productions of the CFG being used.
- As syntactic structure is recognized, the parser either calls corresponding semantic routines directly or builds a syntax tree.
  - CFG (Context-Free Grammar)
  - BNF (Backus-Naur Form)
  - GAA (Grammar Analysis Algorithms)
  - LL, LR, SLR, LALR Parsers
  - YACC

The Structure of a Compiler: Semantic Routines

Semantic Routines:

- Perform two functions
  - Check the static semantics of each construct
  - Do the actual translation
- The heart of a compiler
  - Syntax Directed Translation
  - Semantic Processing Techniques
  - IR (Intermediate Representation)

The Structure of a Compiler: Optimizer

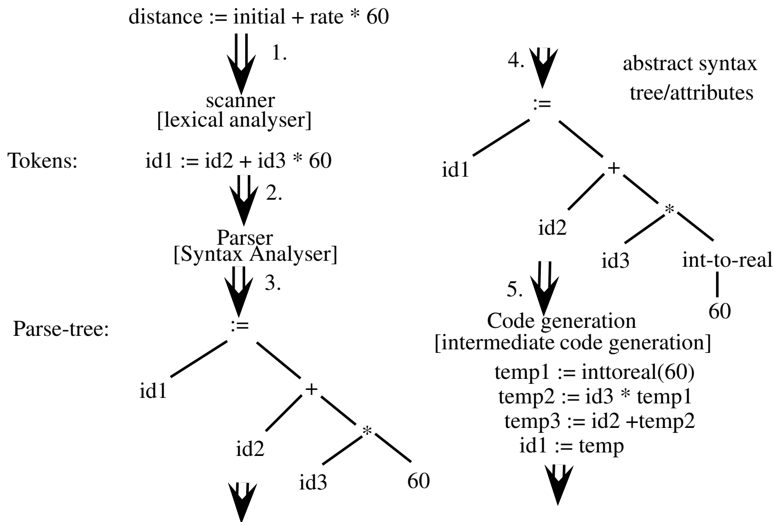
Optimizer :

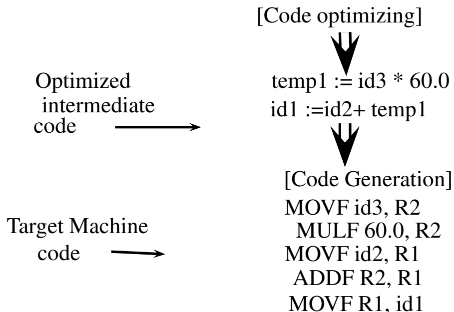
- The IR code generated by the semantic routines is analyzed and transformed into functionally equivalent but improved IR code
- This phase can be very complex and slow
- Peephole optimization
- loop optimization, register allocation, code scheduling
    - Register and Temporary Management
    - Peephole Optimization

Code Generator:

- Interpretive Code Generation
- Generating Code from Tree/Dag
- Grammar-Based Code Generator

The Structure of a Compiler





Symbol Table

1.	Position	....
2.	Initial	....
3.	rate	....
4.	....	....

Tools

Compiler writing tools:

- Compiler generators or compiler-compilers
    1. E.g. scanner and parser generators
    2. Examples : Yacc, Lex

The Syntax and Semantics of Programming Language

- A programming language must include the specification of syntax (structure) and semantics (meaning).
- Syntax typically means the context-free syntax because of the almost universal use of context-free-grammar (CFGs)

  Ex.
- a = b + c is syntactically legal
- b + c = a is illegal

- The semantics of a programming language are commonly divided into two classes:

  Static semantics

- Semantics rules that can be checked at compiled time. Ex. The type and number of a function's arguments

  Runtime semantics

- Semantics rules that can be checked only at run time